

2)

(19) 日本国特許庁 (J P)

(12) 公開特許公報 (A)

(11) 特許出願公開番号

特開2000-20291

(P2000-20291A)

(43) 公開日 平成12年1月21日 (2000.1.21)

(51) Int.Cl. ⁷	識別記号	F I	テーマコード* (参考)
G 0 6 F 9/06	5 3 0	C 0 6 F 9/06	5 3 0 C 5 B 0 4 2
11/28	3 4 0	11/28	5 3 0 C 5 B 0 7 6
			3 4 0 C

審査請求 未請求 請求項の数19 O L (全 19 頁)

(21) 出願番号 特願平10-190815

(22) 出願日 平成10年7月6日 (1998.7.6)

(71) 出願人 000003207

トヨタ自動車株式会社

愛知県豊田市トヨタ町1番地

(72) 発明者 足立 盛保

愛知県豊田市トヨタ町1番地 トヨタ自動車株式会社内

(74) 代理人 100075258

弁理士 吉田 研二 (外2名)

Fターム(参考) 5B042 BB07

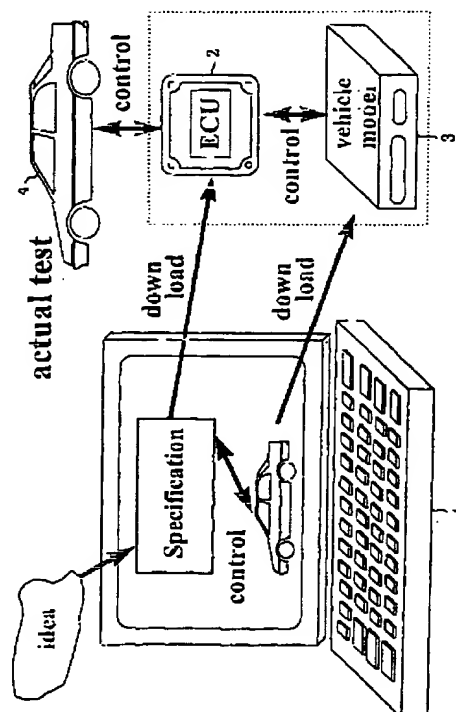
5B076 DD04 DE01 DE03 EC06

(54) 【発明の名称】 車両用プログラム開発支援方法および装置

(57) 【要約】

【課題】 車両制御プログラムの開発における工数を削減する。

【解決手段】 プログラム生成装置1は、データフローチャートおよびステートフローチャートのかたちで入力された制御仕様から車両用Cコードを自動生成する。車両用Cコードは車両ECU2にダウンロードされる。車両モデル装置3を制御対象とする車両ECU2の制御動作がプログラム生成装置1にモニタされ、デバッグ作業が行われる。制御仕様から自動生成した車両用Cコードを使うので確実なデバッグ作業が容易に行われる。また手作業のコーディング作業が不要になる。データフローチャートのシンボルブロックに整数情報をもたせることで、整数情報を利用して整数ロジックの車両用Cコードを自動生成できる。



【特許請求の範囲】

【請求項1】 入力された制御仕様から車両用コードを生成する機能をもつプログラム生成装置を用いて、車両制御プログラムを生成するプログラム生成ステップと、生成された前記車両制御プログラムを車両ECUにダウンロードするダウンロードステップと、前記車両ECUに前記車両制御プログラムを実行させて、前記車両制御プログラムのデバッグを行うデバッグステップと、を含むことを特徴とする車両用プログラム開発支援方法。

【請求項2】 請求項1に記載の車両用プログラム開発支援方法において、前記デバッグステップのデバッグ作業は、前記車両ECUが前記車両制御プログラムを実行した結果を監視する前記プログラム生成装置上で行われることを特徴とする車両用プログラム開発支援方法。

【請求項3】 請求項2に記載の車両用プログラム開発支援方法において、制御対象の車両をモデル化した車両モデル装置に前記車両ECUを接続し、前記車両ECUに前記車両モデル装置を制御させる制御実行ステップと、制御実行中の前記車両ECUおよび前記車両モデル装置を監視する監視ステップと、を含むことを特徴とする車両用プログラム開発支援方法。

【請求項4】 請求項3に記載の車両用プログラム開発支援方法において、入力された車両仕様に基づいて車両モデルを前記プログラム生成装置上で生成するモデル生成ステップと、前記モデル生成ステップで生成された車両モデルを前記車両モデル装置にダウンロードするモデルダウンロードステップと、を含むことを特徴とする車両用プログラム開発支援方法。

【請求項5】 請求項1～4のいずれかに記載の車両用プログラム開発支援方法において、前記車両用コードは、車両ECUで処理される整数ロジックに適合するように一般コードを变形したものであることを特徴とする車両用プログラム開発支援方法。

【請求項6】 請求項5に記載の車両用プログラム開発支援方法において、前記プログラム生成ステップでは、前記プログラム生成装置に入力された整数変換条件に従った車両制御プログラムが生成されることを特徴とする車両用プログラム開発支援方法。

【請求項7】 車両の制御仕様を入力する入力手段と、前記制御仕様に基づいて、車両制御プログラムの車両用コードを生成するプログラム生成手段と、前記車両制御プログラムを外部の車両ECUにダウンロ

ードするダウンロード手段と、前記車両ECUで前記車両制御プログラムを実行した実行結果を出力する出力手段と、を含むことを特徴とする車両用プログラム開発支援装置。

【請求項8】 車両の制御仕様を表すデータフローチャートおよびステートフローチャートを作成するチャート作成機能と、作成されたチャートに基づいて、車両ECUで処理される整数ロジックで構成された車両制御プログラムの車両用コードを生成するプログラムコード生成機能と、を有することを特徴とする車両用プログラム開発支援装置。

【請求項9】 請求項8に記載の車両用プログラム開発支援装置において、物理値に対応する浮動小数点数および浮動小数点数から変換された整数をそれぞれ適用して前記データフローチャートのシミュレーションを行い、浮動小数点数および整数のシミュレーション結果を出力するシミュレーション機能を有することを特徴とする車両用プログラム開発支援装置。

【請求項10】 請求項9に記載の車両用プログラム開発支援装置において、浮動小数点数および整数を適用したときのシミュレーション結果の相違を判定可能なように、整数のシミュレーション結果から浮動小数点数を逆算した結果を表示することを特徴とする車両用プログラム開発支援装置。

【請求項11】 請求項10に記載の車両用プログラム開発支援装置において、前記データフローチャートのブロックシンボルに、浮動小数点数、整数、浮動小数点数から整数への整数変換条件、および該整数変換条件を用いて浮動小数点数から整数を逆算した結果、についての情報をもたせたことを特徴とする車両用プログラム開発支援装置。

【請求項12】 請求項11に記載の車両用プログラム開発支援装置において、前記シミュレーション結果に基づいて前記整数変換条件を調整可能であることを特徴とする車両用プログラム開発支援装置。

【請求項13】 請求項8～12のいずれかに記載の車両用プログラム開発支援装置において、前記ステートフローチャート上で同一階層の複数の前記データフローチャートを実行する順番を定義する優先付け機能を有することを特徴とする車両用プログラム開発支援装置。

【請求項14】 請求項8～13のいずれかに記載の車両用プログラム開発支援装置において、前記データフローチャート内の所望のシンボル接続ラインを選択して所望のラベルを付けるラベリング機能を有し、

該箇所の変数名として前記ラベルを用いた前記車両用コードが生成されることを特徴とする車両用プログラム開発支援装置。

【請求項15】 請求項8～14のいずれかに記載の車両用プログラム開発支援装置において、車両用コード生成の際に、前記データフローチャート内の複数のブロックシンボルに対応する複数の処理をグルーピングするグルーピング機能を有することを特徴とする車両用プログラム開発支援装置。

【請求項16】 請求項15に記載の車両用プログラム開発支援装置において、グルーピングされるブロックシンボルの数を規定する所定のグルーピング制限条件に従ってグルーピングを行うことを特徴とする車両用プログラム開発支援装置。

【請求項17】 請求項15または16のいずれかに記載の車両用プログラム開発支援装置において、前記データフローチャート内の所望のシンボル接続ラインを選択して所望のラベルを付けるラベリング機能を有し、前記ラベルを付けられた箇所が、グルーピングの区切りとされることを特徴とする車両用プログラム開発支援装置。

【請求項18】 請求項8～17のいずれかに記載の車両用プログラム開発支援装置において、前記プログラムコード生成機能により、一般のCコードを車両ECUに適合するように変形した車両用Cコードが生成されることを特徴とする車両用プログラム開発支援装置。

【請求項19】 車両の制御仕様を表すデータフローチャートおよびステートフローチャートを作成するチャート作成機能と、作成されたチャートに基づいて、車両ECUで処理される整数ロジックで構成された車両制御プログラムの車両用コードを生成するプログラムコード生成機能と、をコンピュータに実現させるためプログラムを記録した、車両用プログラム開発支援に用いられるコンピュータ読取可能な記録媒体。

【発明の詳細な説明】

【0001】

【発明の属する技術分野】本発明は、車両用プログラム(vehicle-use program)開発支援方法および装置に関し、特に、開発効率の向上および開発されるプログラムの信頼性の向上に関する。

【0002】

【従来の技術】ビジュアルプログラミング技術を利用した制御システム設計支援統合環境を実現するための技術が進展している。このような背景の下、車両の制御システムの開発にも、シミュレーション機能およびプログラミング機能を備えたソフトウェアをインストールしたコンピュータが利用されるようになってきた。

【0003】システム開発を行うユーザは、シミュレーション機能を利用して制御仕様書(specification)をコンピュータに入力する。コンピュータのディスプレイ上には、ユーザのアイデアを盛り込んだ仕様書が、データフローチャートおよびステートフローチャートといったかたちで書き表される。作成されたチャートに従ったシミュレーション計算が行われ、シミュレーション結果を参考にしてロジックのデバッグ作業が行われる。

【0004】この種のシミュレーションソフトウェアは、作成されたチャートからプログラムコード、例えばCコードを自動的に生成する機能をもつ。しかし、自動生成されるCコードは、冗長で、容量が大きく、そのため、メモリの制約や実行速度に関する制約の厳しい車両制御にそのまま使うのには適していない。また、従来のコード生成方法では、コンピュータが決めた変数名がチャートの各部に割り付けられる結果、生成されたCコードは可読性の低いものとなってしまう。

【0005】そこで、ユーザは、データフローチャートおよびステートフローチャートを参考にして、車両ECU(electric control unit)に適した車両用Cコードを手作業で作成する。特に、一般のCコードと異なり、整数ロジックで構成される専用コードが作成され、これにより、データビット数の観点からも少ないメモリ容量で高速処理が実現される。手作業で作られた車両用Cコードは実際に車載ECUにダウンロードされ、そしてデバッグ作業が行われる。

【0006】

【発明が解決しようとする課題】従来は、上記のように、仕様シミュレーションとプログラムコーディングが全く別の作業であり、それぞれの段階で独立してデバッグ作業を行わなければならなかった。特に、車両用コードのデバッグ作業において不具合が見つかった場合、不具合原因がコーディングのミスにあるのか、不適当なオリジナル仕様書にあるのかを判定するのが困難であった。手作業のコーディングおよびデバッグ作業の困難さは、制御プログラムの信頼性保証にかかる工数を増大させ、さらには、開発期間を引き延ばし、開発コストの増大を招く可能性があった。

【0007】本発明は上記課題に鑑みてなされたものであり、その目的は、信頼性の高い制御プログラムをより短期間で開発できるようにする車両用プログラム開発支援方法および装置を提供することにある。

【0008】

【課題を解決するための手段】(1)上記目的を達成するため、本発明の車両用プログラム開発支援方法によれば、入力された制御仕様から車両用コードを生成する機能をもつプログラム生成装置を用いて、車両制御プログラムが生成される。生成された前記車両制御プログラムは車両ECUにダウンロードされる。そして、前記車両ECUに前記車両制御プログラムを実行させて、前記車

両制御プログラムのデバッグが行われる。このように、本発明によれば、制御仕様から直接的に生成された車両用コードが車両ECUにダウンロードされる。元の制御仕様と車両用コードが共通性をもつので、バグ発生が低減し、少ない作業量で効率よくデバッグ作業を行うことが可能となる。また、コーディング作業における開発者の手作業が大幅に削減される。

【0009】好ましくは、前記デバッグステップのデバッグ作業は、前記車両ECUが前記車両制御プログラムを実行した結果を監視する前記プログラム生成装置上で行われる。制御仕様をもっているプログラム生成装置を利用することで、デバッグ作業が容易になる。

【0010】(2) 本発明の一態様の車両用プログラム開発支援装置は、車両の制御仕様を表すデータフローチャートおよびステートフローチャートを作成するチャート作成機能と、作成されたチャートに基づいて、車両ECUで処理される整数ロジックで構成された車両制御プログラムの車両用コードを生成するプログラムコード生成機能と、を有する。車両ECUでは、メモリサイズや実行速度の制約があるので整数ロジックを用いることが好適である。本態様では、データフローチャートおよびステートフローチャートから整数ロジックのコードを生成することで、車両用コードの自動生成ができる。

【0011】好ましくは、本発明の一態様では、物理値に対応する浮動小数点数および浮動小数点数から変換された整数をそれぞれ適用して前記データフローチャートのシミュレーションを行い、浮動小数点数および整数のシミュレーション結果を出力するシミュレーション機能が設けられる。従来技術では、シミュレーション段階においては浮動小数点数が扱われ、コーディング段階では整数が扱われた。しかし、本発明によれば、シミュレーション段階で浮動小数点数と整数が処理され、両方の処理結果が出力されるので、仕様作成段階で整数ロジックの妥当性を容易に判定することができる。好ましくは、浮動小数点数および整数を適用したときのシミュレーション結果の相違を判定可能なように、整数のシミュレーション結果から浮動小数点数を逆算した結果を表示する。

【0012】また、好ましくは、前記データフローチャートのブロックシンボルに、浮動小数点数、整数、浮動小数点数から整数への整数変換条件、および該整数変換条件を用いて浮動小数点数から整数を逆算した結果、についての情報をもたせる。ブロックシンボルのもつ整数情報を利用して車両用コードが自動生成される。

【0013】(3) 本発明の一態様では、前記ステートフローチャート上で同一階層の複数の前記データフローチャートを実行する順番を定義する優先付け機能が設けられる。これにより、無駄が無く適正な制御プログラムを確実に作成できる。

【0014】また、本発明の一態様では、前記データフ

ローチャート内の所望のシンボル接続ラインを選択して所望のラベルを付けるラベリング機能が設けられる。そして、該当箇所の変数名として前記ラベルを用いた前記車両用コードが生成される。ユーザに分かりやすいラベルを付けることで、可読性の高い車両用コードが生成される。

【0015】また、本発明の一態様では、車両用コード生成の際に、前記データフローチャート内の複数のブロックシンボルに対応する複数の処理をグルーピングするグルーピング機能が設けられる。好ましくは、グルーピングされるブロックシンボルの数を規定する所定のグルーピング制限条件に従ってグルーピングが行われる。また好ましくは、シンボル接続ラインにラベルを付けた箇所が、グルーピングの区切りとされる。このようなグルーピングにより、車両用コードの可読性を向上することができる。

【0016】本発明の車両用コードは、例えば、一般のCコードを車両ECUに適合するように変形した車両用Cコードであるが、ただしCコードには限定されない。本発明の車両ECUは、車両に搭載される任意のECUであり、例えば、エンジンECU、トランスミッションECU、サスペンション制御ECU、ブレーキ制御ECUである。ECUおよび制御対象の車載機器に応じて、プログラム開発に必要な制御仕様および車両仕様が異なることはもちろんである。

【0017】

【発明の実施の形態】以下、本発明の好適な実施の形態(以下、実施形態という)について、図面を参照し説明する。

【0018】図1を参照すると、プログラム生成装置1は、制御仕様の作成機能と、制御仕様のシミュレーション機能と、制御仕様から車両用Cコードを生成する機能を有する。

【0019】プログラム生成装置1はコンピュータで構成され、図2に示すように、CPU10、ROM11、RAM12、キーボード13、ポインティングデバイス14、ディスプレイ15、ハードディスク16、通信回路17、CD-ROMドライブ18を有する。プログラム生成装置1の各種機能を実現するプログラムは、ROM11、ハードディスク16あるいはCD-ROM19に格納されており、CPU10により実行される。入力装置は、キーボード13、ポインティングデバイス14、ディスプレイ15には限定されない。また、CD-ROM19以外の任意のタイプの記録媒体、例えばDVDが用いられてもよい。さらに、ハードディスク16以外の任意のタイプの記憶装置が適用されてもよい。

【0020】ユーザは、キーボード13およびポインティングデバイス14(マウス等)を操作し、自らのアイデアを盛り込んだ仕様書(specification)をディスプレイ15に書き表す。ディスプレイ15には、図3(a)

(b) に示すデータフローチャート(data flowchart、ブロック線図)およびステートフローチャート(state flowchart、state transition chart、状態遷移図)のかたちで制御仕様書が描かれる。データフローチャートが部分的なデータの流れを表し、ステートフローチャートが全体的な制御の流れを表す。

【0021】また、ユーザは、キーボード13およびポインティングデバイス14を操作して車両仕様(諸元)を入力し、これにより、プログラム生成装置1内に仮想的な車両モデルが形成される。具体的には、車両モデルは、車両の種々の運動、動作等を記述した式の集合で構成される。

【0022】ユーザの指示に応え、作成された仕様書(データフローチャートおよびステートフローチャート)のシミュレーションが行われる。図1に示すように、コンピュータ上で制御仕様のロジックが車両モデルを制御する。シミュレーション結果がディスプレイ15に表示され、この段階で必要なデバッグ作業が行われる。デバッグ作業を行いながら、仕様書の修正および改良が行われ、仕様書が完成される。

【0023】仕様書が完成すると、ユーザの指示に従い、仕様書から車両用Cコードが生成される。すなわち、データフローチャートおよびステートフローチャートに対応するCコードを書き込んだファイルが形成される。車両用Cコードは整数ロジックを採用しており、一般のCコードと比較して少ないメモリサイズで高速処理を実現可能に構成されている(後述)。生成された車両用Cコードは、実際の車両ECU2、例えばエンジンECUにダウンロードされる。

【0024】また、プログラム生成装置1から車両モデル装置3へは、車両モデルがダウンロードされる。DSPを含む車両モデル装置3は、高速の割込処理等により、車両ECU2にとっては見かけ上は実際の車両と同様に動作する。開発期間の短縮と効率のよい開発のためには、実際の車両を使ったテストの前にこのようなシミュレーションを行うことが効果的である。

【0025】車両モデル装置3が車両ECU2に接続され、車両ECU2にダウンロードした車両用Cコードが実行される。これにより、現実に近い環境でのシミュレーションが行われる。車両ECU2および車両モデル装

$$x_int = int(x_float - OFFSET) / SLOPE \quad \dots (1)$$

ここで、 x_float は浮動小数点数であり、 x_int は整数であり、OFFSET(またはbias)およびSLOPE(またはLSB)は整数変換条件である。

【0032】(2) 整数ロジックの概要

図4には、車両ECUにおける整数ロジックの概念が示されている。センサ21は、検出した物理量 p に応じた電圧信号 v をA/D変換器22に送る。A/D変換器22は電圧信号 v を整数データ P に変換し、車両ECU2に送る。車両ECU2は、整数ロジックを用いて、検出

置3の動作がプログラム生成装置1により通信回路17を使ってモニタされ、そして、プログラム生成装置1上でデバッグ作業が行われる。例えば、不適当な制御動作が発見されると、その原因がデータフローチャートおよびステートフローチャートを使って調べられる。コンピュータ上で不具合箇所の修正が行われ、修正後のロジックが検証される。

【0026】デバッグ作業の終了後、車両ECU2は車両4に搭載され、実車テスト(actual test)が行われる。ここでは、実際の車両の非線形要因等に起因する修正必要箇所が検出され、最終的なプログラム修正作業が行われる。

【0027】以下、プログラム生成装置1についてより詳細に説明する。

【0028】[制御仕様の入力・作成機能(整数ロジック)]従来は、制御仕様を入力する段階では、物理値をそのまま表す浮動小数点数(floating point number)を処理するチャートが作成される。そして、浮動小数点数を計算したシミュレーション結果に基づいて、制御仕様の妥当性が判断される。従来機能によりチャートから自動的に生成されるCコードも浮動小数点数を含む。

【0029】しかし、車両制御では、低コストおよび処理速度の観点から、ECUのCPUの能力を最大限に引き出すことが要求される。高速回転するエンジンなどの機器の制御においては、この要求が顕著である。このような要求に応えるため、車両ECU用のCコードに整数ロジック(固定小数点数(fixed point number))を適用し、一般Cコードとはデータビット数を異ならせることが好適である。この点が、車両用Cコードと自動生成される一般Cコードとの大きな相違点であり、開発者による手作業での車両用Cコード作成を余儀なくする理由であった。

【0030】本実施形態では、後述する整数ブロックを導入することにより、制御仕様の作成段階から整数ロジックを考慮できるようにし、そして、制御仕様から整数ロジックの車両用Cコードを自動生成可能にする。

【0031】(1) 浮動小数点数から整数への変換
浮動小数点数から整数への変換は、下式(1)に従って行われる。;

【数1】

信号の整数データ P から制御パラメータの整数データ Q を得る。整数データは、例えば、16bitの符号無し整数(unsigned integer)である。整数データ Q は制御信号に変換され、制御対象のアクチュエータ23に出力される。そして、アクチュエータ23が動作して物理量 q が生じる。

【0033】図4において、整数ロジックは、物理値を対象とするオリジナルの制御ロジックと同等の結果をもたらさなければならない。車両ECU2の入出力部のS

LOPE、OFFSETは、それぞれ lp 、 lq 、 op 、 oq である。整数ロジックには実際には複数段階の演算処理が設けられ、中間過程の幾つものSLOPE、OFFSETが設けられる。これらのSLOPE、OFFSETは、メモリサイズおよび実行時間の観点から整数ロジックを最適化するように設定されるべきである。

【0034】例えば、ユーザが作成した制御仕様(ロジック)の一部に、図5に示すような伝達関数があったとする。整数ロジックの構成にあたり、伝達関数の7つの項のすべてについて同じSLOPEを使ったとする。第1項と第7項では係数が大幅に違うため、第1項に適したSLOPEを採用すると、第7項の数値は常にゼロになってしまう。

【0035】一般的に、小さすぎるSLOPEはオーバーフローの原因になる。また、大きすぎるSLOPEは結果的に一部のセンサの入力の無視を招くことになる。さらに、不適当なSLOPE設定が原因で制御ロジックが冗長成分を含むこともあり得る。このようなことを回避するため、適切な整数変換条件を容易に設定可能にすることが求められる。

【0036】(3) 整数ブロック

データフローチャート上で適切な整数ロジックを構成し、適切な整数変換条件を容易に設定できるようにするために、本実施形態では、その特徴として図6の整数ブロック(Integer Block)を導入する。整数ブロックは、データフローチャート内の一つのブロックシンボルである。ユーザは、整数ブロックのシンボルをデータフローチャートに書き込むことができる。

【0037】整数ブロックは、図6に示すように、(1) integer (整数)、(2) SLOPE (LSB)、(3) OFFSET (bias)、(4) float (浮動小数点数)、(5) Data Type、(6) "integer \times SLOPE + OFFSET" の6種の情報をもつ。(6)は、integerからfloatを逆算した結果(逆算float)である。

【0038】図7および図8を参照し、かけ算を例にして、整数ブロックの機能を説明する。図7のデータフローチャートにおいて、かけ算ブロックには、入力データ1、2とともに、それぞれ、SLOPEおよびOFFSETが入力される。かけ算ブロックは、これらのデータから上記の6つの情報を求めて出力する。

【0039】図8には、かけ算ブロックの処理が式で示されている。入力値 x 、 y がそれぞれoffset1、2およびslope1、2を用いて整数に変換され、これら2つの整数の積が出力される(integer出力)。一方、float出力は、 xy である。第1式を xy と比較すると、出力側のSLOPEおよびOFFSETが分かる(図8、第2式および第3式)。さらに、逆算float(=integer \times SLOPE_{out}+OFFSET_{out})が出力される。

【0040】ユーザは、図7の表示を見て、シミュレーション結果を知ることができる。floatを見ることで、基本の制御ロジックが正しいか否かが分かる。さらに、floatと逆算されたfloatが比較される。両者の差(0.1)が許容範囲内であれば、整数ロジックが適当であることが分かる。一方、両者の差が許容範囲外であれば、整数ロジック、すなわち整数変換条件であるslope1、2の設定が不適当であることが分かる。そこで、適当な結果が得られるまでslope1、2が調整される。

【0041】図7および図8には、制御仕様の一部のみが例示されている。しかし、実際の制御仕様は、連結された一連の多数のブロックで構成される。本実施形態によれば、制御過程の各ブロックでSLOPEの検証と調整を行うことができる。例えばあるブロックでfloatとintegerのずれが大きければ、着目ブロック以前のいずれかのブロックの整数変換条件が調整される。

【0042】従って、制御過程の全体において、floatとintegerが実質的に同等の意味をもつことができる。言い換えれば、後段のブロックに行くに従ってfloatとintegerとのずれが拡大するというような事態の発生が回避される。また、不適当なSLOPEの設定によってあるセンサの入力が無視されるというような事態の発生も回避される。

【0043】図8では、その第3式に示されるように、出力側のOFFSETが、入力値 x 、 y を含んでいる。この影響が無視できない場合には、ユーザは、図9の変形処理を選択できる。図9では、その第1式に示されるように、かけ算の前に整数に"offset/slope"を足すことにより、出力側のoffsetの項がゼロになる。

【0044】また、上記説明では、かけ算を例にして整数ブロックの機能を述べたが、他の種類の演算に関しても同様の機能が設けられる。図10～図16は、整数ロジックをサポートする各種の整数ブロックを示している。なお、図10に示すように、このツールでは、整数ブロックに情報 $y(5) \sim y(8)$ が付加されているが、基本的な情報 $y(1) \sim y(4)$ は上記と同様である。また、図10～図16では、LSBがslopeに相当している。

【0045】以上のように、本実施形態では、整数ブロックの導入により、チャート形式の制御仕様書が、整数ロジックを表現する。理論的な制御ロジック(実数演算)と整数ロジック(整数演算)とを比較しながら適切な整数変換条件を設定できるので、バグの少ない適正な整数ロジックを容易に作成できる。作成された制御仕様書は、それ自体がコンピュータ上でシミュレーション可能である。さらに、制御仕様書の整数ロジック部分を抽出してCコードを生成すれば、車両用Cコードを自動的

に生成できる。制御仕様書と車両用Cコードが共通性をもつので、コード生成段階でのバグ発生が低減し、また、デバッグ等のプログラム検証も容易になる。

【0046】[優先付け機能(演算順序の規定)]複数のモジュールまたは演算式の順序を規定する場合、一般には、図17に示すように、各モジュールまたは演算式がトリガー付きサブシステム(TriggeredSubsystem)とされ、それらのサブシステムに対してステートフローチャートからファンクションコール(function-call)が行われる。

【0047】本実施形態では、さらに、図18の優先付け機能が設けられている。ユーザは、データフローチャートに優先順位を表す番号を付ける。各モジュールまたは演算式は、優先付けされたサブシステム(Prioritized Subsystem)となる(1_Prior_Subsystem、2_Prior_Subsystem)。演算は、サブシステムの名前の順に行われる。

【0048】この優先付け機能によれば、同一階層のデータフローチャートの演算順序の規定が容易にできるという利点が得られる。また、図19に示すように、生成されるCコードが従来より簡単になるという利点があり、これは可読性およびメモリ容量の面から好適である。

【0049】[Cコード生成(ラベリング)]Cコード生成は、ユーザの指示に応じて行われる。前述のように、整数ブロックの適用により、データフローチャートの各ブロックが整数情報をもつ。従って、一連のブロック群を整数演算で辿ることができる。整数演算を書き出すことにより、整数ロジックで構成された車両用Cコードが生成される。具体的には、図7の例を参照すると、チャートの段階でかけ算ブロックに入出力される情報群の中から、整数演算(2つの整数の入力と1つの整数の出力)が抽出される。

【0050】既存のソフトウェア製品を変形して利用する場合には、チャートから一般Cコードを自動生成するツールと、Cコードを書き込む雛形ツールとを用意し、そして、整数ロジックに適するように雛形を変形すればよい。このとき、一般Cコードにはない専用コマンド、および一般Cコードを変形した専用コマンド等に関する処置も施される。

【0051】図20は、データフローチャートから生成されるCコードの一例を示している。通常のCコード生成方法では、図20に示すように、コンピュータが決めた変数名が各部に割り付けられる。そのため、生成されたCコードは、人間にとっては非常に読みづらいものになる。

【0052】このように一般的なCコード自動生成方法では、可読性はあまり考慮されていない。主としてCコードとチャートの一致が求められているからである。しかし、車両システム開発では、実際のECUを使ったテストで動作検証が行われる。このとき、開発者がCコー

ドを読みとらなければならない機会が多い。従って、車両用Cコードの自動生成では、可読性が高いことが求められる。

【0053】本実施形態では、可読性向上のために、図21に示すラベリング機能が設けられている。ユーザは、データフローチャート内のブロックシンボルの接続ラインをクリックする。選択した接続ラインに所望のラベルが付けられる。図21では、例として、ラベル“t_x”“t_y”“t_z”“t_xyz”が付けられているが、実際には、該当するモジュール名やセンサ名などの分かりやすいラベルを付けることが好適である。Cコード生成の際には、ラベルが付けられた箇所の変数名として該当ラベルが使われる。その結果、図21に示すように、読みやすいCコードが生成される。

【0054】なお、図21の例ではすべての接続ラインにラベルが付けられている。しかし、ユーザは、Cコードを読みやすくするために適当な所望のラインにラベルを付ければよい。ラベリングされないラインについては、コンピュータが割り付けた変数名がそのまま使用される。

【0055】[Cコード生成におけるグループ化]

(1) 通常のCコード生成では、データフローチャートの一つ一つのブロックに個別の変数名が付けられる。そして、各ブロックに対応して、一つの変数を導き出す式が生成される。そのため、生成されたCコードは、多数の変数と式を含み、冗長で読みづらい。

【0056】そこで、本実施形態では、可読性の向上のためにグルーピング機能が設けられている。図22を参照すると、データフローチャートでは、入出力の間に2つのブロックがある。グループ化が行われていない左側のCコードでは、ブロック数に応じた2つの式が生成されている。一方、グループ化された右側のCコードでは、2つのブロックの演算がまとめられ、一つの式で表されている。人間がCコードを読む上では不要な変数名が削減されており、従って、読みやすいCコードが生成されている。

【0057】ただし、多数のブロックをグルーピングすると、一つの式が長くなりすぎて、かえって可読性を低下させる可能性がある。そこで、一つのグループに含まれるブロックの数を規定する制限条件を設けることが好適である。ここでは、ブロックの数の最大値を2つとする。従って、Cコードの一つの式は、最大2段階の演算を行う。

【0058】図23および図24を参照し、グルーピング処理を説明する。Cコード生成の際、データフローチャートの各ブロックにIDが付けられ、そのIDが変数名として使われる。IDとユーザがつけたラベルとを関連づける変数名(信号名)データベース(図23

(a))が作成される。ラベルが付けられていない場合には、IDがそのまま使用される。

【0059】図24において、まず、入力信号を調べる関数によって、着目するブロックの入力信号のIDが調べられる(S11)。そして、式のデータベース(図23(b))を参照して、各入力信号について、グルーピング条件(最大2回)が成立しているか否かが判断される(S12)。条件未成立(2回未満)であれば、数式のデータベースが検索される(S13)。そして、入力IDに対応する式が採用される。一方、条件が成立すれば、変数名のデータベースが検索され(S14)、該当するラベルが変数名として採用される。ラベルがなければIDが採用される。

【0060】S13の式またはS14の変数名を用いて、着目ブロックの式が作成される(S15)。作成した式に関してグルーピング条件が成立したか否かが判断される(S16)。条件未成立であれば、次のブロックでの処理のために、式のデータベースに、S15で作った式が登録される(S17)。グルーピング条件が成立すれば、S15で作った式がファイルに書き出される(S18)。図22では、2つのブロックをまとめた式が出力される。

【0061】図25を参照して、上記のグルーピング処理の具体例を説明する。入力信号はs1およびs2である。入力信号s1について式のデータベースを検索すると、式 $(x+1)$ が得られる。この式のグルーピング回数はまだ1回である(条件未成立)。そこで、式 $(x+1)$ を入力信号として採用する。次に、入力信号s2についての検索を行うと、入力信号s2のグルーピング回数は2回であることが分かる。そこで、変数のデータベースが検索され、ラベルx2が入力信号として採用される。従って、着目しているブロック内の演算は、 $(x+1) \times x2$ である(かけ算)。

【0062】次に、変数名のデータベースから出力信号名txが検索される。入力信号s1に関して2回目のグルーピングを行っているので、グルーピング条件が成立している。そこで、 $t_x = (x+1) \times x2$ が出力される。もしこの段階でまだグルーピング条件が成立していなければ、作成された式はデータベースに登録される。

【0063】(2)グルーピング処理では、さらに、チャート内でラベルが付けられた場所は、必ずグループの区切りとされる。ユーザが付けたラベルはすべてCコードの演算式に含まれる。ユーザがチャート内の適当な場所にラベルを付ければ、チャートの構造に対応する構成をもつCコードが生成される。このような処理により、自動生成されるCコードがさらに読みやすくなる。

【0064】例えば、ユーザが、データフローチャートの最初と最後にラベルを付けたとする。この場合、最初と最後の変数を明瞭に示し、かつ、処理過程を適当な間隔で区切られた、可読性の高いCコードが自動生成される。

【0065】以上より、本実施形態によれば、Cコード

生成の際に適切なグルーピングを行うことにより、自動生成されるCコードの可読性をさらに高めることが可能となる。また、変数名を減らしてメモリを節約することができる。

【0066】[車両ECUへのダウンロード] 図1を参照して述べたように、プログラム生成装置1は、ユーザの指示に応え、自動生成した車両用Cコードを車両ECU2にダウンロードする。車両ECU2は車両用Cコードを実行して車両モデル装置3(DSP)を制御する。プログラム生成装置1は、制御の実行の様子をモニタし、ディスプレイ上に適当なかたちで表示する。図26は、車両ECU2にエンジンECUを適用した場合のモニタ画面例を示している。ユーザは、プログラム生成装置1を操作して、車両用Cコードの動作検証およびデバッグ作業を行う。

【0067】本実施形態では、バグの少ない適正な車両用Cコードが生成されている。車両用Cコードと元の制御ロジックが共通している。さらに、車両用Cコードの可読性が高い。従って、デバッグ作業を効率よく容易に行うことができる。

【0068】

【発明の効果】以上に説明したように、本発明によれば、ユーザが入力した制御仕様に対応する車両用コードが自動的に生成されるので、コーディング作業の工数が大幅に削減される。入力された制御仕様は、それ自体がコンピュータ上でシミュレーション可能なものであり、かつ、車両用コードに変換して車両ECUでも使用可能なものであり、従って、デバッグ作業の工数の大幅削減が可能となる。その結果、信頼性保証のためにかかる工数が削減され、さらに、開発期間の短縮および開発コストの低減が可能になる。特に、車両制御プログラムの肥大化と複雑化という背景の下では、仕様書作成からロジックデバッグまでを支援する本発明の利点が顕著に得られる。

【0069】また、本発明によれば、ラベリングやグルーピングを行う適切なCコード生成処理により、可読性の高い車両用コードが自動生成される。これにより、デバッグ作業等の工数をさらに削減することができる。

【図面の簡単な説明】

【図1】 本発明の実施形態の全体構成を示す図である。

【図2】 プログラム生成装置の構成を示すブロック図である。

【図3】 制御仕様を表すデータフローチャートおよびステートフローチャートを示す図である。

【図4】 車両ECUにおける整数ロジックの概念を示す図である。

【図5】 整数ロジックの一部の例を示す図である。

【図6】 整数ブロックの構成を示す図である。

【図7】 整数ブロックをかけ算に適用した例を示す図

である。

【図8】 図7のかけ算ブロックの処理を示す図である。

【図9】 図8の処理の変形例を示す図である。

【図10】 整数ロジックのサポートツールを示す図である。

【図11】 整数ロジックのサポートツールを示す図である。

【図12】 整数ロジックのサポートツールを示す図である。

【図13】 整数ロジックのサポートツールを示す図である。

【図14】 整数ロジックのサポートツールを示す図である。

【図15】 整数ロジックのサポートツールを示す図である。

【図16】 整数ロジックのサポートツールを示す図である。

【図17】 演算順序の規定方法を示す図である。

【図18】 演算順序の規定方法であって本実施形態の優先付け機能を示す図である。

【図19】 図17および図18の結果として生成され

るCコードを示す図である。

【図20】 データフローチャートから生成されるCコードの例を示す図である。

【図21】 図20と同様のデータフローチャートから生成されるCコードであって、本実施形態のラベリングが行われたCコードを示す図である。

【図22】 Cコード生成におけるグルーピング処理を示す図である。

【図23】 Cコード生成過程で作成される変数名データベースおよび式のデータベースを示す図である。

【図24】 図23のデータベースを利用したグルーピング処理を示す図である。

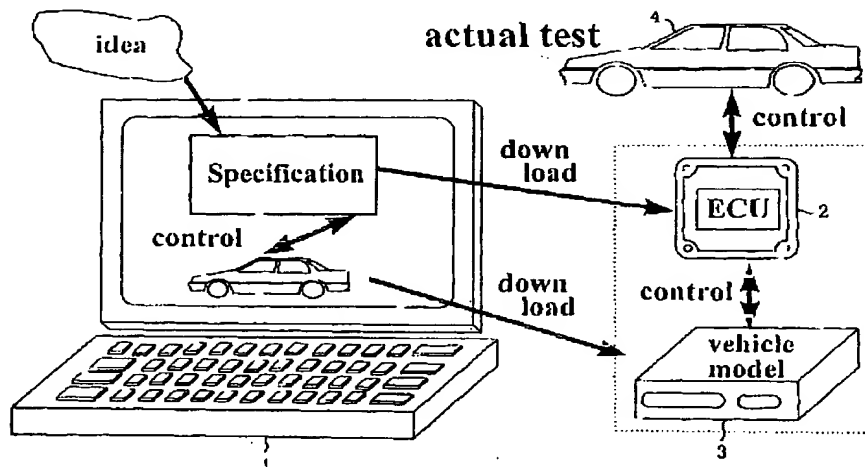
【図25】 グルーピング処理の具体例を示す図である。

【図26】 プログラム生成装置がエンジン制御をモニタするときの表示画面例を示す図である。

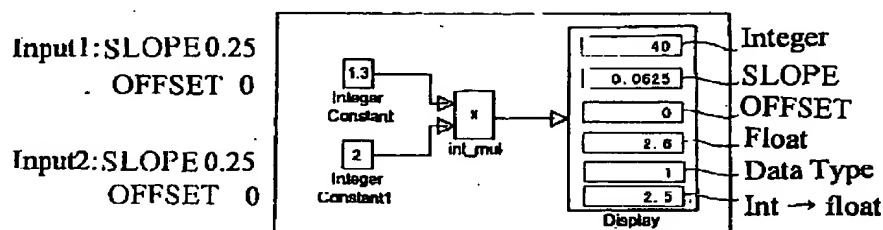
【符号の説明】

1 プログラム生成装置、2 車両ECU、3 車両モデル装置、10 CPU、11 ROM、12 RAM、13 キーボード、14 ポインティングデバイス、15 ディスプレイ、16 ハードディスク、17 通信回路、19 CD-ROM。

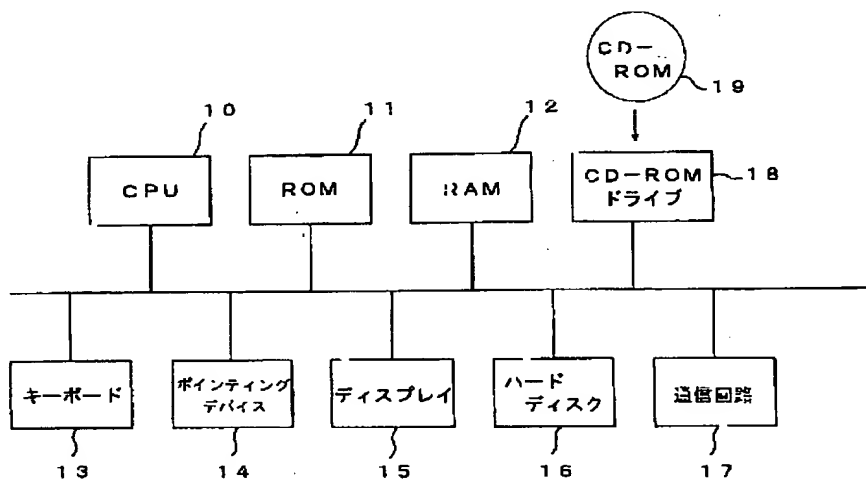
【図1】



【図7】

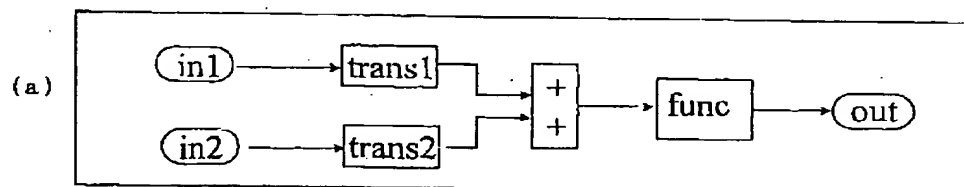


【図2】

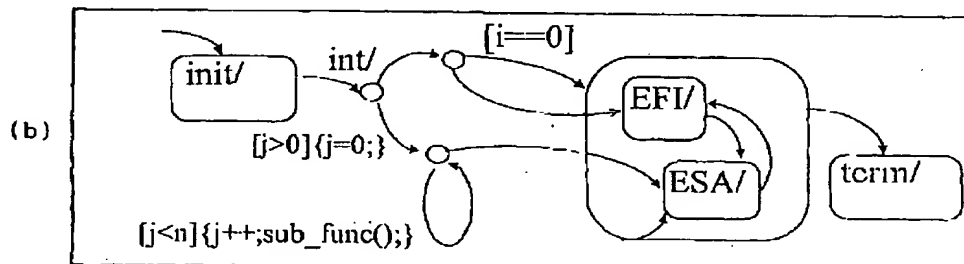


【図3】

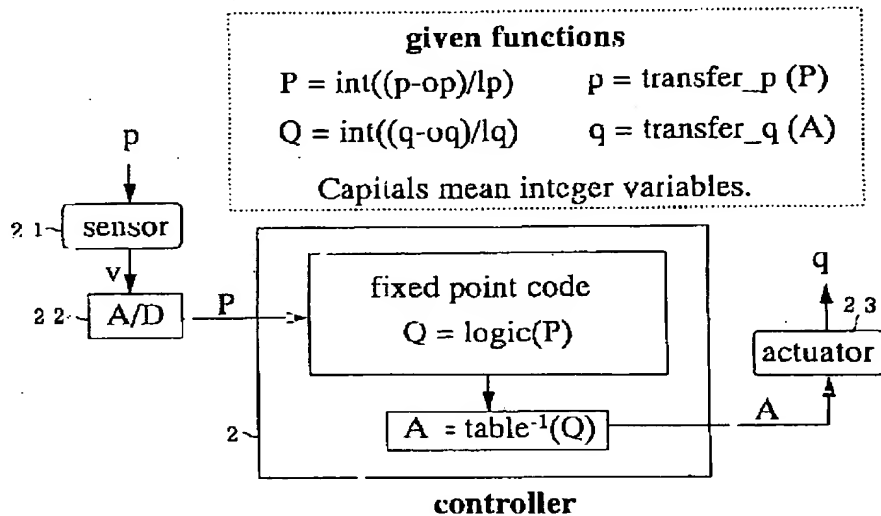
データフローチャート (DATA FLOWCHART)



ステートフローチャート (STATE FLOWCHART)



【図4】

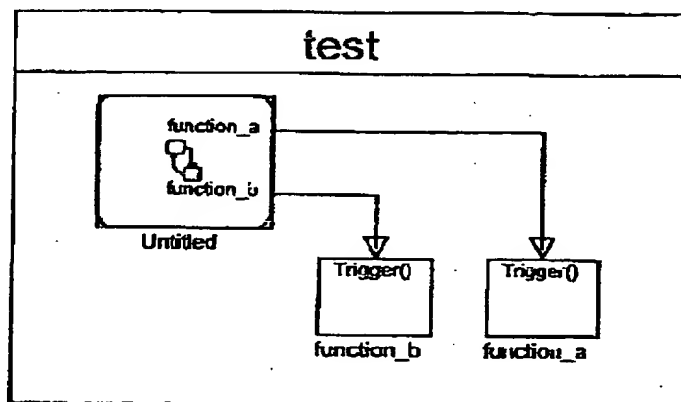
INTEGER LOGIC FEATURE

【図5】

$$\begin{aligned}
 y(k) = & 0.7888 y(k-1) + \\
 & 0.1784 y(k-2) - \\
 & 0.1000 y(k-3) - \\
 & 0.0010 u(k) + \\
 & 0.0150 u(k-1) - \\
 & 0.0040 u(k-2) - \\
 & 0.0020 u(k-3)
 \end{aligned}$$

example of
floating point formula

【図17】



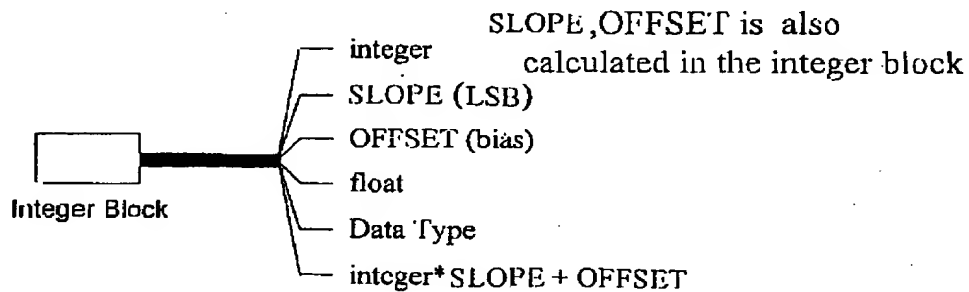
【図6】

Integer Block's Signal

Translation formula

float \rightarrow integer

$$x_{int} = (x_{float} - OFFSET) / SLOPE$$



【図8】

整数ブロックのかけ算 (1)

multiply

$$\frac{\text{input1}}{\text{slope1}} \times \frac{\text{input2}}{\text{slope2}}$$

$$= \frac{\text{Output}}{xy - \text{offset1} * y - x * \text{offset2} + \text{offset1} * \text{offset2}} \div \text{slope1} * \text{slope2}$$

Output SLOPE : slope1*slope2

Output OFFSET : offset1*y + x*offset2 - offset1*offset2

【図9】

整数ブロックのかけ算 (2)

multiply

$$\frac{\text{input1}}{\text{slope1}} + \frac{\text{offset1}}{\text{slope1}} \times \left(\frac{\text{input2}}{\text{slope2}} + \frac{\text{offset2}}{\text{slope2}} \right) = \frac{\text{Output}}{\text{slope1} * \text{slope2}}$$

$X_{\text{int}} \qquad Y_{\text{int}}$

specifies SLOPE : slope_o → output

specified OFFSET : offset_o → output

OFF1_{int} : offset1/slope1, OFF2_{int} : offset2/slope2

multiplied integer value :

$$((X_{\text{int}} + \text{OFF1}_{\text{int}}) * (Y_{\text{int}} + \text{OFF2}_{\text{int}}) * \text{slope1} * \text{slope2} - \text{offset}_o) / \text{slope}_o$$

【図10】

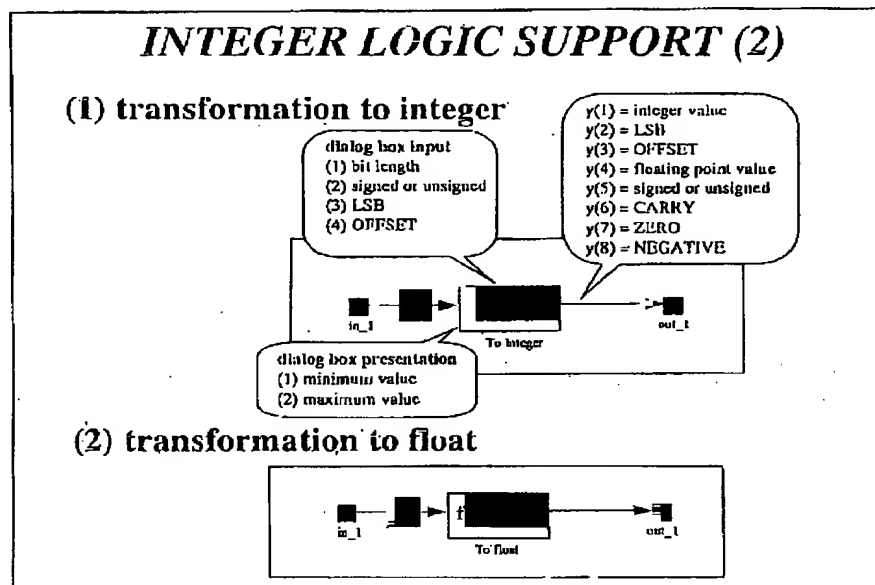
INTEGER LOGIC SUPPORT (1)**(1) All integer lines are vectors composed of**

- | | |
|-----------------------------|---------------------------|
| y(1) : integer value | y(5) = signed or unsigned |
| y(2) = LSB | y(6) = CARRY |
| y(3) = OFFSET | y(7) = ZERO |
| y(4) = floating point value | y(8) = NEGATIVE |

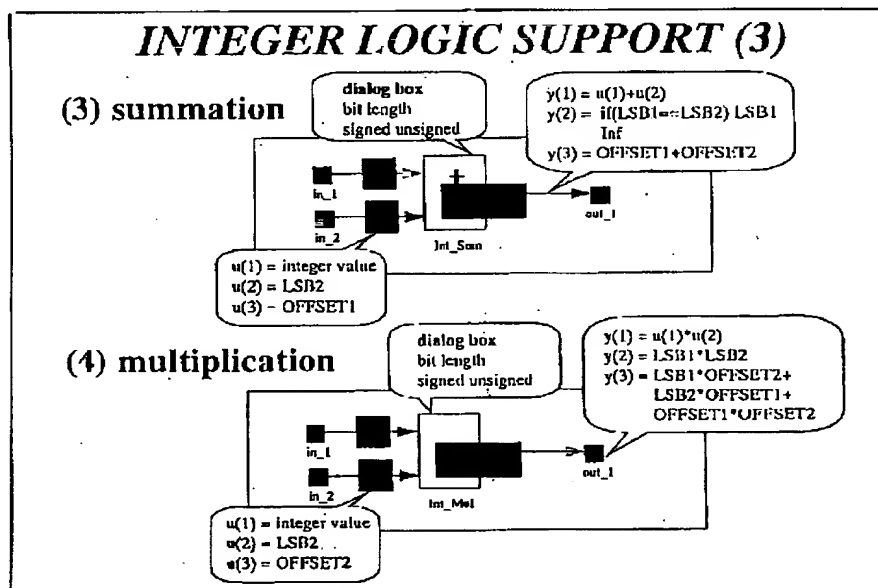
(2) The integer logic support tool has the following blocks.

- | | |
|-------------------------------|-------------------------|
| (1) transformation to integer | (9) two D table look up |
| (2) transformation to float | (10) split |
| (3) summation | (11) unit delay |
| (4) multiplication | (12) integer scope |
| (5) division | |
| (6) residual | |
| (7) shift | |
| (8) one D table look up | |

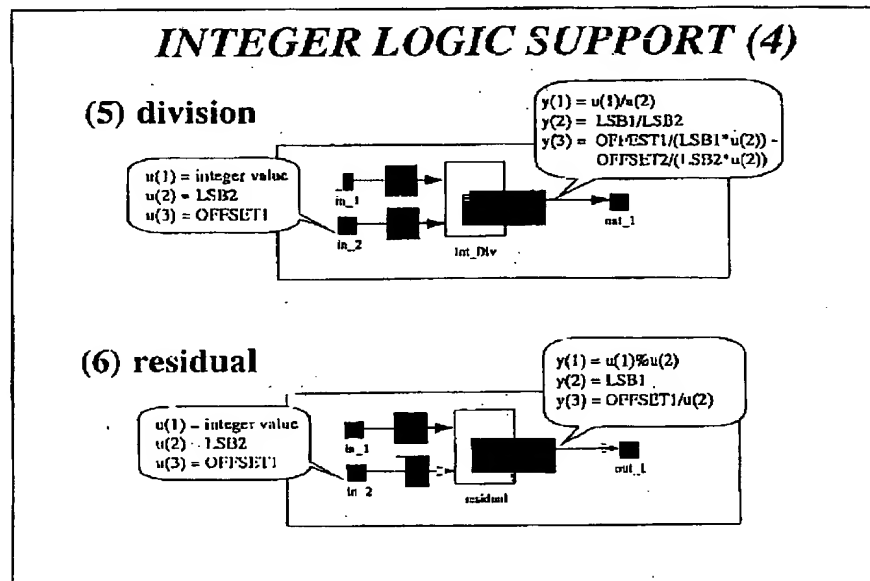
【図11】



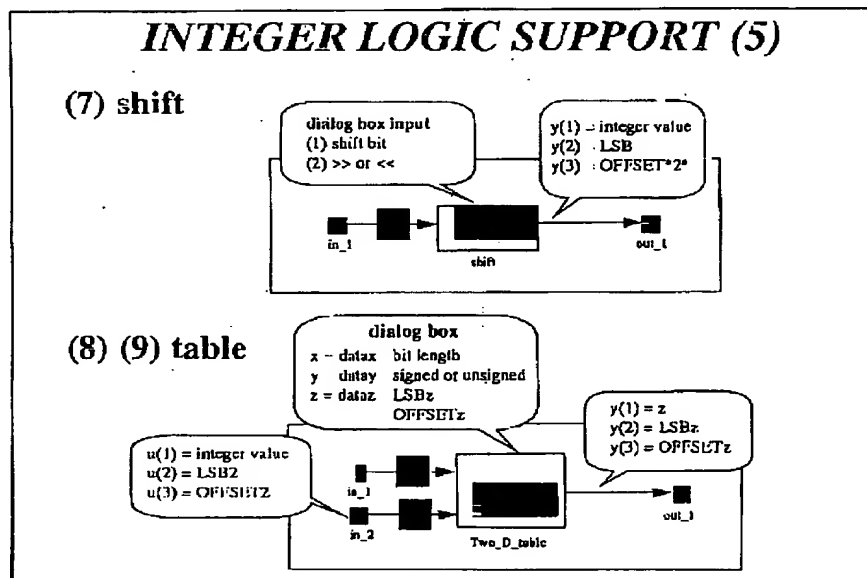
【図12】



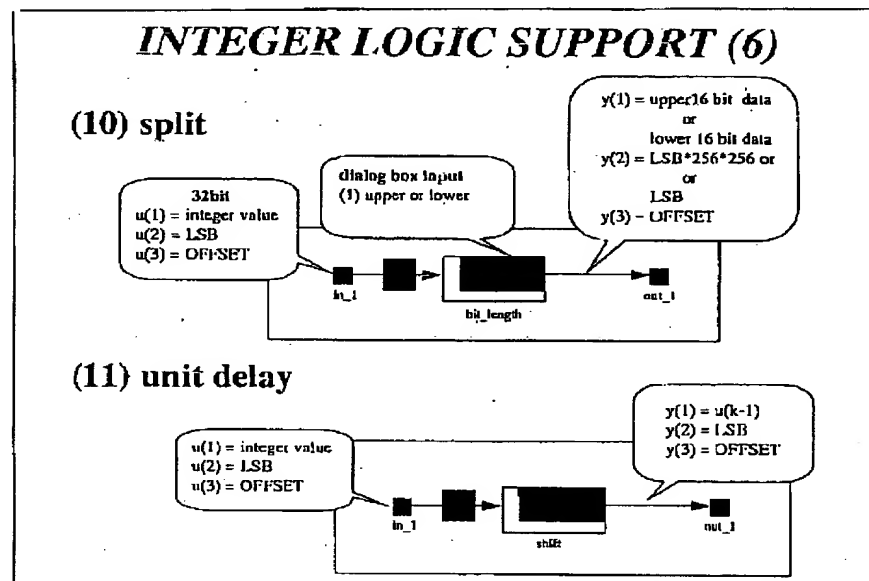
【図13】



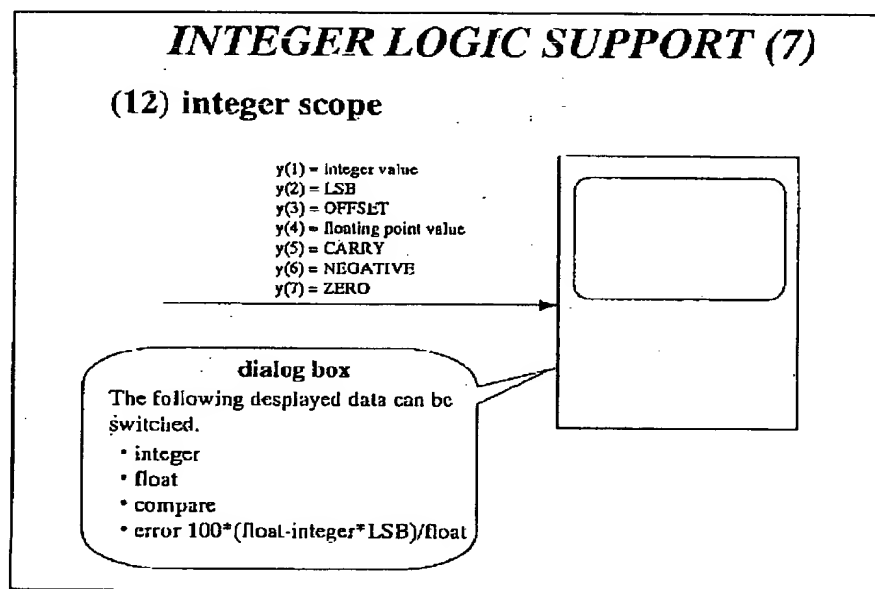
【図14】



【図15】

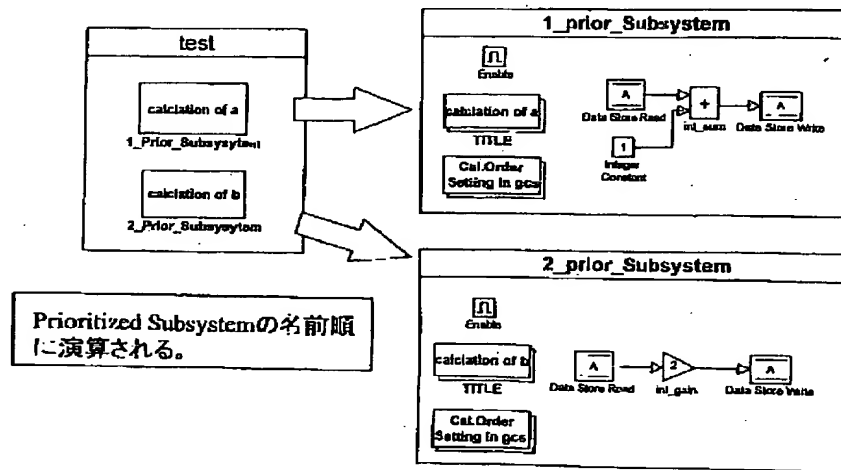


【図16】

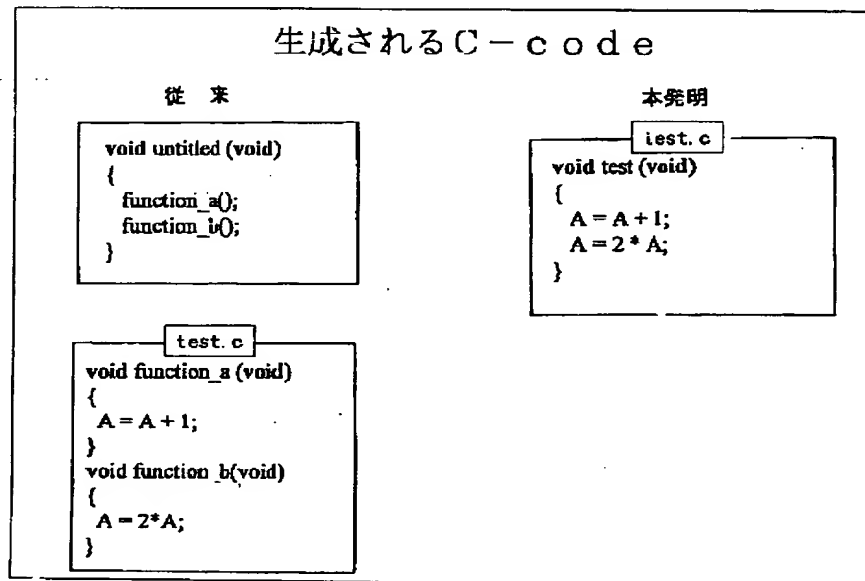


【図18】

(Prioritized Subsystem)

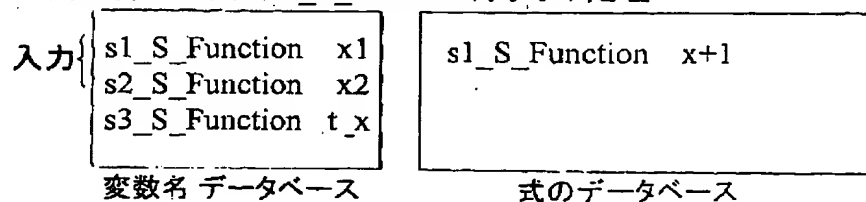


【図19】



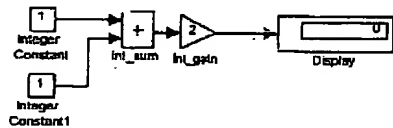
【図25】

掛け算ブロック(s3_S_Function)内での処理



【図20】

C-code生成



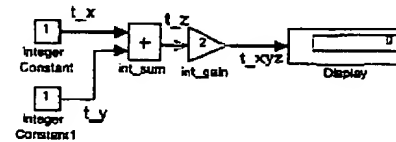
```
void untitled(void)
{
    sl6 s1_S_Function;
    sl6 s2_S_Function;
    sl6 s4_S_Function;
    sl6 s3_S_Function;

    /* int_sum : s4_S_Function */
    s4_S_Function = s1_S_Function + s2_S_Function;

    /* int_gain : s3_S_Function */
    s3_S_Function = (sl6)(2*s4_S_Function);

    /* (no update to perform in root model) */
}
```

【図21】

C-code生成
(ラベリング機能)

```
void untitled(void)
{
    sl6 t_x;
    sl6 t_y;
    sl6 t_z;
    sl6 t_xyz;

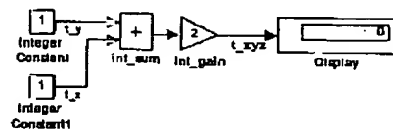
    /* int_sum : s4_S_Function */
    t_z = t_x + t_y;

    /* int_gain : s3_S_Function */
    t_xyz = (sl6)(2*t_z);

    /* (no update to perform in root model) */
}
```

【図22】

C-code生成 (グルーピング)



グルーピング無し

```
void untitled(void)
{
    sl6 t_x;
    sl6 t_y;
    sl6 s4_S_Function;
    sl6 t_xyz;

    s4_S_Function = t_x + t_y;
    t_xyz = (sl6)(2*s4_S_Function);
}
```

グルーピング

```
void untitled(void)
{
    sl6 t_x;
    sl6 t_y;
    sl6 t_xyz;

    /* int_gain : s3_S_Function */
    t_xyz = (sl6)(2*t_x + t_y);
}
```

【図23】

グルーピング

(a)

ID	Signal Label
s1_S_Function	t_x

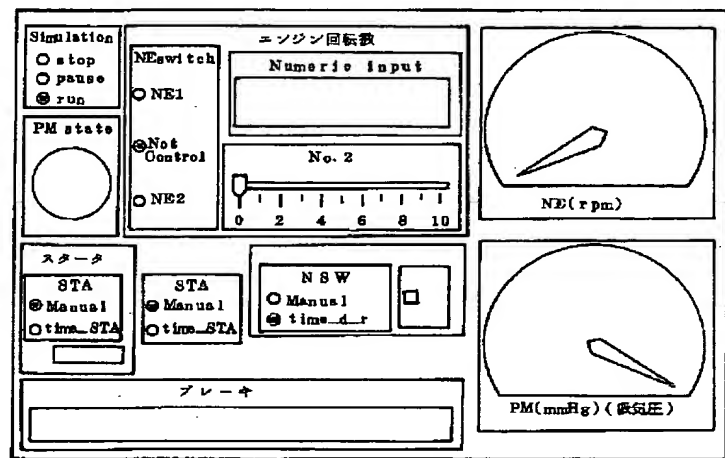
変数名 データベース

(b)

ID	式
s1_S_Function	$x1 * x2$

式のデータベース

【図26】



【図24】

